RESEARCH PLAN PROPOSAL

Generation of Test Data and Test Cases for Software Testing: A Genetic Algorithm Approach

For registration to
Doctor of Philosophy

IN THE FACULTY OF COMPUTER SCIENCE

to

THE IIS UNIVERSITY, JAIPUR

Submitted By:

Shveta Parnami

ICG/2010/11466

Under the Supervision of:

Prof.(Dr.) K.S.Sharma
Advisor, The IIS University

Prof. Swati V. Chande
Principal MCA, ISIM

Department of CS & IT

April, 2011

Generation of Test Data and Test Cases for Software Testing: A Genetic Algorithm Approach

1. Introduction:

1.1 Software Testing

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. It is laborious, costly and time consuming task: it spends almost 50% of software system development resources (Srivastava and Tai-hoon, 2009). Software testing is performed for defect detection and reliability estimation. The goal of software testing is to design a set of minimal number of test cases and test data such that it reveals as many faults as possible (Srivastava and Tai-hoon, 2009),(Faezeh et. al, 2010)). In software testing process each test case has an identity and is associated with a set of inputs and a list of expected outputs (Last et. al, 2005). Test data generation is the process of identifying a set of program input data that satisfies a given testing criterion (Ghiduk and Girgis, 2010).

Testing is a static and dynamic activity. Black box and White box testing are two categories of software testing. It includes unit, interface, functionality, performance, usability, security and operability testing of software. Software testing is executed by conducting a program developed with test inputs and comparing the observed output with the expected one. It is difficult to test the whole software, therefore selective parts of the software are considered for testing. As the input space of the Software Under Test (SUT) might be very large, testing has to be conducted with a representative subset of test cases. Creating relevant subset of test cases during software testing is the most critical activity. The test cases which are used to examine the SUT must possess an ability to expose faults; they must constitute representative subset of possible inputs (Srivastava and Tai-hoon, 2009). The quality and the significance of the overall test are directly affected by the set of test cases that are used during testing. An effective set of test cases is one that has a high probability of detecting faults (Gupta and Rohil, 2008). This requirement of effective test cases demands the generation of 'Good' automated test cases. Test cases are created by using test data. Another critical task in automation of software testing is generation of test data to satisfy a given adequacy criterion. Test data generation technique and application of a test data adequacy criterion decide the 'Better' test data (Ghiduk and Girgis, 2010). There is need to explore these aspects of test data generation in order to increase the degree of automation and efficiency of software testing.

1.2 Genetic Algorithm

Genetic Algorithms (GAs) are adaptive heuristic search techniques premised on the evolutionary ideas of natural and genetic selection. The basic concept of GAs is designed to simulate processes in evolution system. Genetic algorithms represent an intelligent exploitation of a random search within a defined search space to solve a problem. GAs have been widely studied, experimented and applied in many fields in the engineering worlds. Genetic algorithms are based on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation- inducing operators, such as mutation and recombination. GAs are useful and work efficiently when the search space is large, complex and poorly understood, when domain knowledge is scarce or expert knowledge is difficult to encode. It is also useful when there is a need to narrow the search space and incase of failure of traditional search methods.

1.2.1 Simple Genetic Algorithm

As an example, a simple genetic algorithm is given below:

```
{
initialize population;
evaluate population;
        while TerminationCriteriaNotSatisfied
        {
                select parents for reproduction;
                perform recombination and mutation;
                evaluate population;
        }
}
```

Genetic Algorithms begin with a set of initial individuals as the first generation, which are sampled at random from the problem domain. The algorithms are developed to perform a series of operations that transform the present generation into a new, fitter generation. Each individual in each generation is evaluated with a fitness function. Based on the evaluation, the evolution of the individuals may approach the optimal solution. The most common operations of genetic algorithms are designed to produce efficient solution for the target problem. These primary operations include:

a) Reproduction: This operation assigns the reproduction probability to each individual based on the output of the fitness function. The individual with a higher ranking is given a greater probability for reproduction. As a result, the fitter individuals are allowed a better survival chance from one generation to the next.

b) Crossover: This operation is used to produce the descendants that make up the next generation. This operation involves the following crossbreeding procedures:
i) Randomly select two individuals as a couple from the parent generation.
ii) Randomly select a position of the genes, corresponding to this couple, as the crossover point. Thus, each gene is divided into two parts.
iii) Exchange the first parts of both the genes corresponding to the couple.
iv) Add the two resulted individuals to the next generation.

c) Mutation: This operation picks up a gene at random and changes its state according to the mutation probability. The purpose of the mutation operation is to maintain the diversity in a generation to prevent premature convergence to a local optimal solution. The mutation probability is given intuitively since there is no definite way to determine the mutation probability.

Upon completion of crossover processing and mutation operations, there will be an original parent population and a new offspring population. A fitness function should be devised to determine which of these parents and offsprings can be survived into the next generation. After performing the fitness function, these parents and offsprings are filtered and a new generation is formed. These operations are iterated until the expected goal is achieved. Genetic algorithms guarantee high probability of improving the quality of the individuals over several generations (Nirpal and Kale, 2010).

1.3 Genetic Algorithm in Software Testing

Test cases and test data generation is the key problem in software testing and its automation improves the efficiency and effectiveness and lowers the high cost of software testing (Nirpal and Kale, 2010). Generation of test data using random, symbolic and dynamic approach is not enough to generate adequate amount of test data. Other problems, like non-recognition of occurrences of infinite loops and inefficiency to generate test data for complex programs makes these techniques unsuitable for generating test data. Therefore there is need for generating test data using search based technique (Ghiduk and

Girgis, 2010). In addition to these there is need of generating test cases that concentrate on error prone areas of code (Berndt et. al, 2003).

Researches have been conducted for generating test cases from the structural approach of testing by examining branch and statement coverage, branch testing and path testing using genetic algorithm. Also for the functional testing, the genetic algorithm is applied for automating test cases, where the program is treated as a black box and extracted from the code the necessary information to calculate the objective function and measuring the time it takes in the process (Berndt et. al, 2003). Genetic algorithm is used to automatically generate test cases that provide good coverage in terms of path it tests or visits within the user interface of tested application. The genetic algorithm is used to convert the task of test case generation into an optimal problem. One of the genetic algorithm applications in software testing is generation of suitable test data. (Nirpal and Kale, 2010)

The application of Genetic Algorithm in Software Testing is an emerging area of research that brings about the cross fertilization of ideas across two domains (Srivastava and Kale, 2009). Genetic Algorithm can be used to generate test cases while ensuring that the generated test cases are not redundant (Rajappa et. al, 2008). This maximizes the test coverage for the generated test cases. In order to carry out the effectiveness of the test cases and test data the quantification, measurement and the perfect modeling is required which is done by using the appropriate suite of software test metrics. The test metrics are used to measure the number, complexity, quality and coverage of generated test cases and test data to make the testing more reliable and quantifiable.

2. Review of Literature:

Srivastava and Tai-hoon, (2009) applied the Genetic Algorithm technique to find the most critical paths for improving software testing efficiency. The authors present a method for optimizing software testing efficiency by identifying the most critical path clusters in a program. This is done by developing variable length Genetic Algorithms that optimize and select the software path clusters, which are weighted in accordance with the criticality of the path. The approach used by the authors is a Weighted Control Flow Graph to test data generation using Genetic Algorithm. Path testing searches the program domain for suitable test cases that cover every possible path in the software under test. Path testing selects a subset of paths to execute and find test data to cover it. The authors conclude that Genetic Algorithm techniques can be applied for finding the most critical paths for improving software testing efficiency and the Genetic Algorithms also outperform the exhaustive search and local search techniques. The experiments conducted by the authors were based on small examples.

The concept of dominance relations between the nodes of Control Flow Graph to reduce the software testing cost has been proposed by Ghiduk and Girgis, (2008). A new fitness function is defined using dominance relationship to evaluate the generated test data. Experiments have been carried out to evaluate the effectiveness of the proposed GA technique as compared to the Random Testing (RT) technique, and to evaluate the effectiveness of the new fitness function and the technique used to reduce the cost of software testing. The results showed that the proposed GA technique outperformed the RT technique. The proposed GA and RT techniques were applied to object oriented C++ programs. Testing on other type of programs, like structure oriented program, should be done for wide acceptability of the results.

Yang et. al., (2009) presented an approach of generating test data for a specific single path based on genetic algorithms. A similarity between the target path and execution path with sub path overlapped is taken as the fitness value to evaluate the individuals of a population and drive GA to search the appropriate solutions. The authors conducted several experiments to examine the effectiveness of the designed fitness function, and evaluated the performance of the function with regards to its convergence

ability and consumed time. Results show that the function performs better as compared with the other two typical fitness functions for the specific paths employed by the authors.

Michael et. al., (1997) proposed that the combinatorial optimization techniques when used with Genetic algorithms solve problems involving the simultaneous satisfaction of many constraints like, condition based decision coverage. The authors have performed experiments for comparing random test data generation with GA by applying genetic search. The genetic search outperformed random test generation, but the authors' experiment did not include the implementation of path selection heuristic for the generation of test data for programs with procedures.

Nirpal and Kale, (2010) have compared the software test data for automatic path coverage using genetic algorithm with Yong Chen approach (Chen Yong, 2009) of generating test data for path testing. They proved that the genetic algorithm approach outperforms the Yong Chen approach. The genetic algorithm is found useful in reducing the time required for lengthy testing by generating the meaningful test cases for path testing. The genetic algorithm is required to be build for structural testing for reduce execution time by generating more suitable test cases.

Faezeh et. al, (2005) focused on the use of independent path to reduce time and on precisely monitoring the execution trace of the program. Genetic algorithm is applied with improved parameters for test cases designed to better detect bugs of tested program. The genetic algorithm based tester fulfills test criteria by manner of evolutionary computation. Genetic Algorithm method with dynamic fitness function and stopping criterion is used for effective testing and low cost identification of infeasible path. The approach used suffers from the disadvantage about dynamic aspect of testing, as the stopping criteria used can't specify actual number of generations, i.e. in some cases, the tester is exited based on waiting time, while the stopping criterion is not satisfied.

Last et. al., (2005) stated that Fuzzy-based Age Extension of Genetic Algorithm [FAexGA] is more efficient to generate black box test cases than Simple Genetic Algorithm because the probability of finding the error with the former approach is much more as compared to the latter, which results in saving a lot of resources for the testing team. The number of distinct solutions produced by FAexGA is significantly higher, which is useful for investigation and identification of the error itself by the software programmers. There is need to apply the proposed methodology to test real programs and to develop evaluation functions for the evolved test cases.

Genetic Algorithm approach has been used by Gupta and Rohil, (2008) to generate test cases for Object Oriented Software where statements are represented in the form of a tree. The test cases are encoded using various strategies and objective functions. Test cases for testing object oriented software include test programs which create and manipulate objects in order to achieve a certain test goal. The approach described by them facilitates the automatic generation of object oriented test program using genetic algorithms. The approach has been used to generate the test cases for Java classes to prove the concept.

Rajappa et. al., (2008) proposed graph theory based genetic approach to generate test cases for software testing. In this approach the directed graph of all the intermediate states of the system for the expected behavior is created. The base population of genetic algorithm is generated by creating a population of all the nodes of the graph. A pair of nodes referred to as parents are then selected from the population to perform crossover and mutation on them to obtain the optimum child nodes. The process is continued until all the nodes are covered. This process is followed for the generation of test case in the real time system. The technique is more concrete in case of network testing or any other system testing where the predictive model based tests are not optimized to produce the output.

As proposed by Berndt et. al, (2003) Genetic algorithm is used to breed software test cases. The Genetic algorithm includes a fossil record that records past organism, allowing any current fitness calculation to be influenced by the past generations. Factors developed for fitness functions are novelty, proximity and severity. Novelty is a measure of the uniqueness of a particular test case, Proximity is the measure of closeness to other test cases that resulted in system failures and Severity is the measure of seriousness of system error. Interplay of these factors produces complex search behavior.

Rauf et. al, (2010) have presented a Graphical User Interface (GUI) testing and coverage analysis technique entered on genetic algorithms in order to exploit the event driven nature of GUI. Event-flow graph technique is used in the field of automated GUI testing. Just as the control-flow graph, GUI that represents all possible execution paths in a program, event-flow model, in the same way, represents all promising progressions of events that can be executed on the GUI. Genetic algorithm searches for the best possible test parameter combinations according to predefined test criterion, like coverage function that measures how much of the automatically generated optimization parameters satisfy the given test criterion.

3. Motivation/Justification and Relevance:

In the light of the literature reviewed, it is observed that testing takes a large portion of the software project resources. Reduction in cost and time at this stage will be of great help for cost effective software development process. Manual testing is slow and expensive, so there is a need to generate non redundant test cases and test data. In many areas of safety-critical software development, the ability to achieve test coverage of code is considered vital. Test data generation is quite labor-intensive and expensive in software testing. The automation of test process may result in significant reduction in the cost of software development. The automation of test cases that concentrate on the error prone areas will help in improving the quality of the software. Test data is often generated manually as such demand for automatic test data generation is high. There has also been an increase in the demand for coverage assessment tools elsewhere in the software development community. This increases the demand for test data generation tools. No powerful test generation tools are commercially available today.

Genetic algorithms are one of the best ways to solve a problem about which only a little knowledge is available. Genetic algorithms use the principles of selection and evolution to produce several solutions to a given problem. Genetic algorithms tend to thrive in an environment in which there is a very large set of candidate solutions and in which the search space is uneven and has many hills and valleys.

Michael et. al., (1997) proved that Genetic search visibly outperforms random test generation. Srivastava and Tai-hoon, (2009) has proved that the Genetic Algorithms also outperform the exhaustive search and local search techniques. Genetic Algorithm can therefore, be used independently for selecting appropriate test data and test cases and solve any problem. Moreover it is an emerging field and has tremendous scope.

4. Objectives:

The objectives of the present research are:

1) To identify and analyze the software testing parameters.
2) To study the existing automated test data and test cases used for software testing.

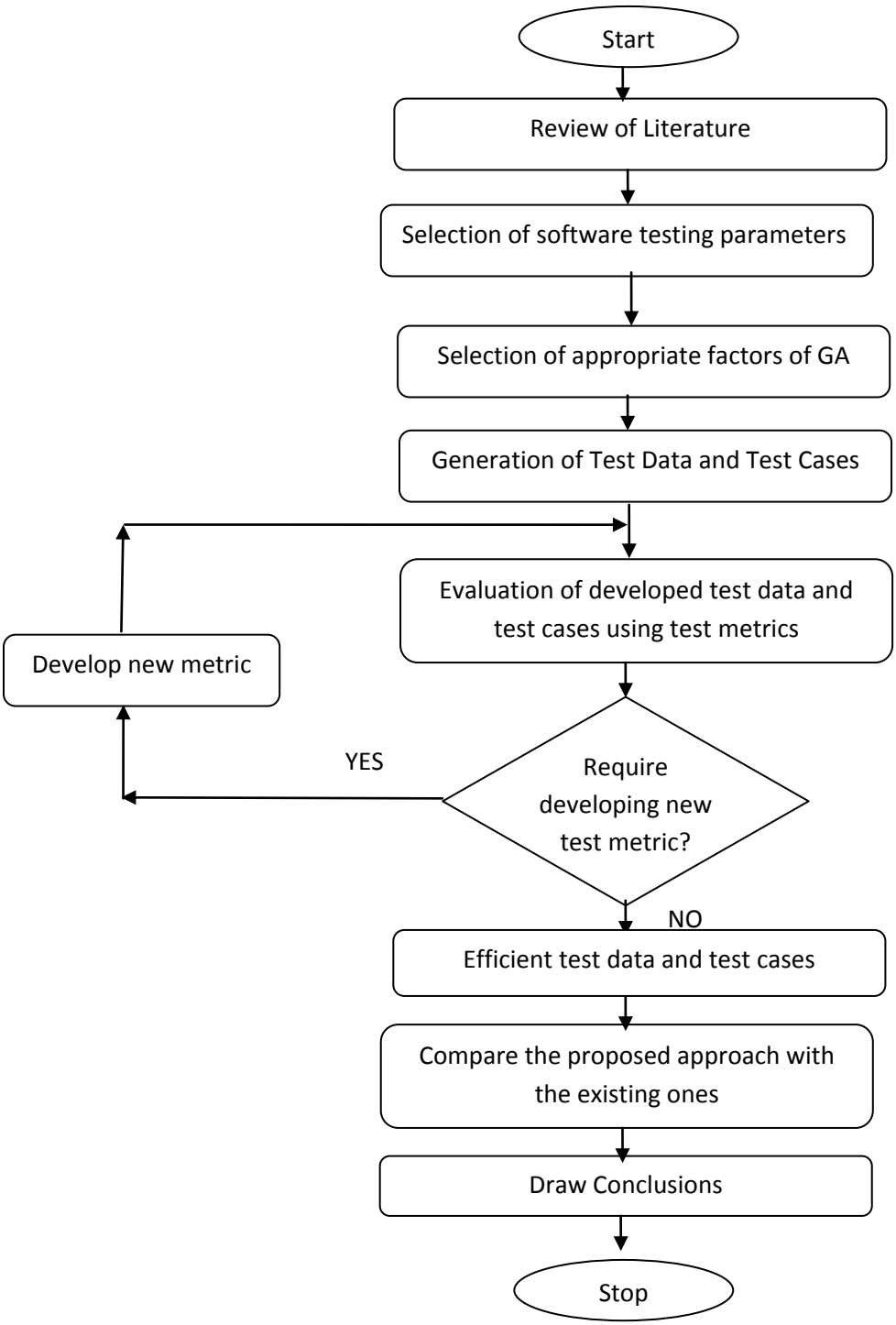
3) To analyze Genetic Algorithm for improving/developing test data and test cases.
4) To study the basic nature of existing genetic transformations used for software testing and its applications for generating the efficient software test cases.
5) To improve upon the existing test data and test cases or generate new ones for software testing, using suitable genetic algorithm operators.
6) To study and improvise the existing test metrics for measuring the number, quality, complexity and coverage of generated test data and test cases.
7) To compare the efficiency and effectiveness of the improvised / the newly developed methods with the existing software testing approaches.

5. Methodology:

In order to achieve the above mentioned objectives the following methodology will be followed:

1) To review the uses of genetic algorithm in software testing, we propose to study corresponding research papers.
2) Identify the parameters to improve the software testing methods on the basis of their dynamic and static nature.
3) Explore the factors of genetic algorithm responsible for developing efficient test data and test cases.
4) Design the new / refine the existing genetic algorithm approach to improve testing methods.
5) Evaluate the effectiveness of such approach on actual case studies.
6) Evaluate the efficiency of generated test data and test cases using test metrics.
7) Refine the existing / develop new test metric to make testing more effective.
8) Comparison of the performance and effectiveness of the newly designed approach with the existing ones.
9) On the basis of comparison the conclusions will be drawn.
10) Presentation of new results in the form of papers and thesis.

6. Flow Chart:


Start
Review of Literature
Selection of software testing parameters
Selection of appropriate factors of GA
Generation of Test Data and Test Cases
Evaluation of developed test data and test cases using test metrics
Develop new metric
YES
Require developing new test metric?
NO
Efficient test data and test cases
Compare the proposed approach with the existing ones
Draw Conclusions
Stop

7. Tools and Techniques:

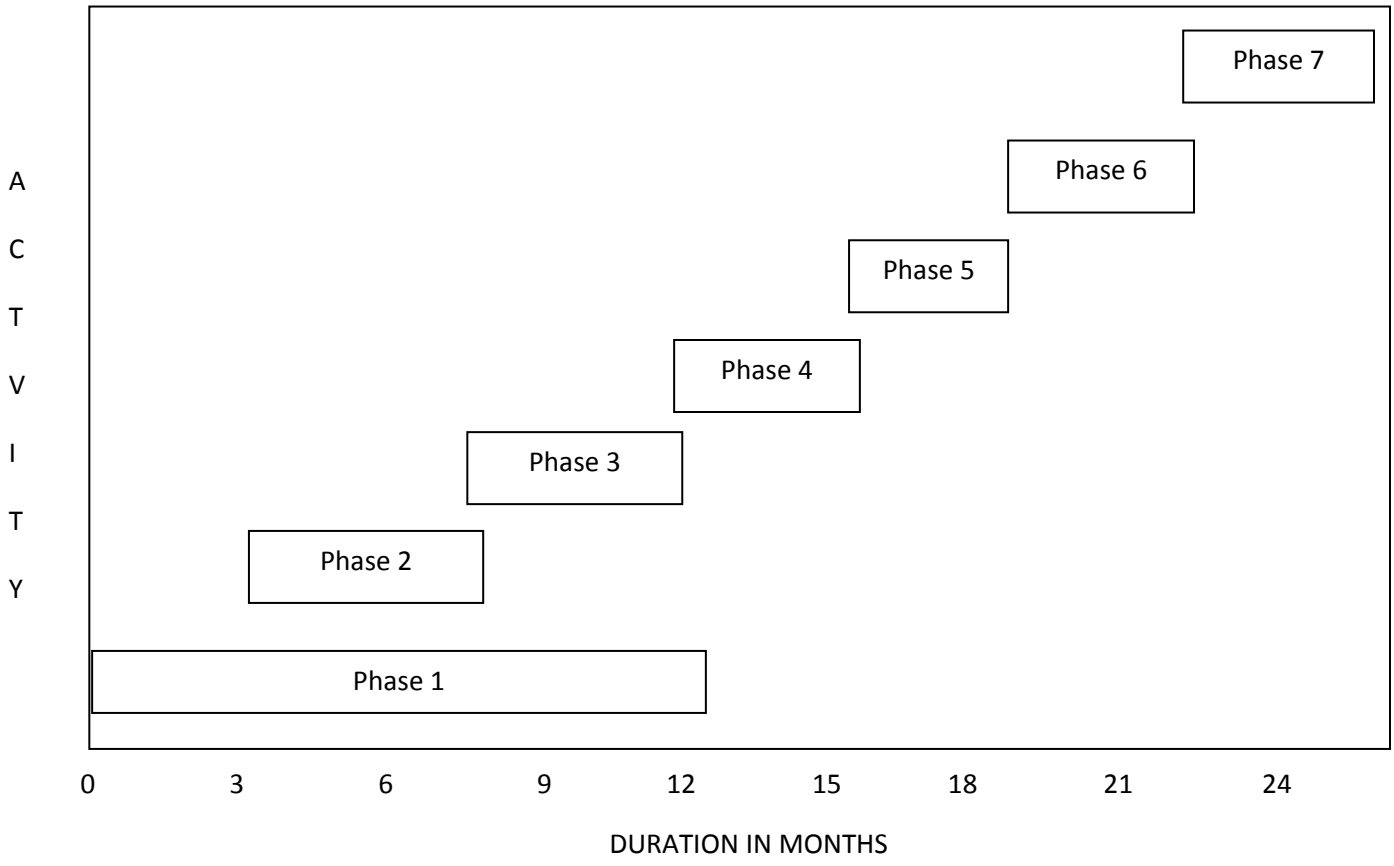
The automation of test data and test cases will be done with the help of Genetic Algorithm approach. Open source automated testing tools will be used to create, manage and run tests for testing types like functional testing, unit testing, coverage testing and object oriented testing. Other tools needed as per requirement of the proposed research, may be used from time to time.

8. Work Plan:

The research plan is divided into following seven phases:

Phase	Activitity	Approximate time (in months)
1.	Literature survey on software testing and genetic algorithm	8-12
2.	Identification of software testing parameters	3-5
3.	Identification of genetic algorithm factors needed for automation of test data and test cases	4-5
4.	Assessment of the effect of reviewed Genetic Algorithm factors on test data and test cases	3-4
5.	Develop new / refine existing methods for improvement in test data and test cases generation	3-4
6.	Implementation, comparison with existing techniques and drawing conclusions	4-5
7.	Paper/ Thesis Writing	2-4

The timeline chart for research plan is as follow:



9. References:

1. Aljahdali S.H , Ghiduk A.S and Telbemy M.(2010), The limitations of Genetic algorithms in software testing, *International Conference on Computer Systems and Applications*, held at Hammamet from 16th May to 19th May 2010, ISBN: 978-1-4244-7716-6, pp: 1-7.

2. Berndt, D.; Fisher, J.; Johnson, L.; Pinglikar, J. and Watkins, A.(2003), Breeding Software Test Cases With Genetic Algorithms, *Proceeding of the 36th Annual Hawaii International Conference on System Sciences*, Track 9, Volume 9, ISBN: 0-7695-1874-5, pp.10.

3. Bouchachia A(2007), An Immune Genetic Algorithm for Software Test Data Generation, 7th *International Conference on Hybrid Intelligent Systems*, Kaiserslauteru, Germany, 17th -19th September, pp.84-89.

4. Chen Yong, Zhong Yong, Tingting Shi1, Liu Jingyong, (2009), "Comparison of Two Fitness Functions for GA-based Path-Oriented Test Data Generation" , *Fifth International Conference on Natural Computation*, IEEE 2009.

5. Faezeh, S. Babamir, Esmaeil Amini, S. Mehrdad Babamir, Ali Norouzi and Berk Burak Ustundag(2010), Genetic Algorithm and Software Testing based on Independent Path Concept, *International Conference on Genetic and Evolutionary Methods-GEM'10, The 2010 World Congress in Computer Science, Computer Engineering and Applied Science*, Las Vegas, Nevada, USA, July 2010.

6. Ghiduk, Ahmed S and Girgis, Moheb R.(2010), Using Genetic Algorithms and Dominance Concepts for Generating Reduced Test Data, *Informatica (Slovenia)*, Volume 34, Number 3, pp.377-385.

7. Gupta, N.K. and Rohil M.K.,(2008), Using Genetic Algorithm for Unit Testing of Object Oriented Software, *International Conference on Emerging Trends in Engineering and Technology*, 16th -18th July 2008, Nagpur, Maharastra, ISBN: 978-0-7695-3267-7, pp. 308-313.

8. Haichang Gao, Boqin Feng and Li Zhu(2005), A Kind of SAaGA Hybrid Meta-heuristic algorithm for the automatic Test Data Generation, *International Conference on Neural Network's and Brain*, pp.111-114.

9. Jin-Cherng Lin and Pu-Lin Yeh, (2001), Automatic Test Data Generation for Path Testing Using Genetic Algorithms, *International Journal of Information Sciences*, January 2001, New York, USA, Volume 131, Issue 1-4,

10. Last Mark, Shay Eyal and Abraham K,(2005), Effective Black-Box Testing with Genetic Algorithms, *Haifa Verification Conference*, pp.134-148.

11. Michael, C.C., McGraw, G.E., Schatz, M.A. and Walton, C.C.,(1997), Genetic algorithms for dynamic test data generation, *International Conference on Automated Software Engineering*, 1st – 5th November 1997, Incline Village, NV, USA, ISBN: 0-8186-7961-1, pp. 307-308.

12. Nirpal, Premal B and Kale K.V. (2010), Comparison of Software Test Data for Automatic Path Coverage Using Genetic Algorithm, *Internal Journal of Computer Science and Engineering Technology*, ISSN:2229-3345 Vol. 1, Issue 1.

13. Rajappa V. , Biradar A.and Panda S.(2008), Effective Software Test Case Generation Using Genetic Algorithm Based Graph Theory, *First International Conference on Emerging Trends in Engineering & Technology,* pp.298-303.

14. Rauf A., Anwar S., Jaffer M.A.and Shahid A.A.(2010), *Automated GUI Test Coverage Analysis Using GA*, 7th International Conference on Information Technology New Generations, pp. 1057-1062.

15. Srivastava P.R and Tai-hoon Kim(2009), Application of Genetic Algorithm in Software Testing, *International Journal of Software Engineering and its Applications*, October 2009, Vol. 3, No. 4.

16. Yang Cao, Chunhua Hu and Luming Li(2009), An Approach to Generate Software Test Data for a Specific Path Automatically with Genetic Algorithm, *International Conference on Reliability, Maintainability and Safety*, pp.888-892.